John von Neumann Institute for Computing

Ring Algorithms on Heterogeneous Windows-based Clusters with Various Message Passing Environments

A. Clematis, A. Corana

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 195-202, 2006.

http://www.fz-juelich.de/nic-series/volume33

Ring algorithms on heterogeneous Windows-based clusters with various message passing environments

Andrea Clematis [a], Angelo Corana [b]

[a]IMATI-CNR, Via De Marini 6, 16149 Genova, Italy

[b]IEIIT-CNR, Via De Marini 6, 16149 Genova, Italy

The aim of the present work is the development of efficient and self-adaptive ring algorithms on heterogenous Windows-based clusters. We show that a virtual ring of processes, with a number of processes on each node proportional to its relative speed, greatly reduces load imbalance and allows to achieve good performance even on highly heterogeneous systems.

As test application we consider the computation of long- and short-range interactions.

Two different implementations of MPI for Windows are considered (MPICH and MPICH2) and some comparisons with PVM are also performed. The analysis is quite general and can be applied to similar problems. From the algorithm analysis we obtain both a full computer simulator of ring applications and some simplified indices of performance, useful to quickly adapt the application to a given platform.

1. Introduction

In recent years clusters of PCs for parallel computing have become very popular owing to several favourable characteristics, namely good cost to performance ratio, availability, flexibility.

The increasing power of nodes and speed of interconnecting network [1] allow to build very powerful machines, able to compete with traditional high performance computers. Recently, PC clusters and/or networks of PCs are evolving towards the emerging Grid architecture (desktop and global grids) or they can be viewed as components of a larger grid. Although most of network-based parallel computing systems run some Unix O.S. (in particular Linux), the systems based on Windows O.S. are also interesting, since they allow to exploit for parallel applications the great number of Windows machines available in various organizations, both scientific, business and industrial.

In this work we consider the development and the analysis of efficient and self-adaptive ring algorithms on heterogeneous clusters (i.e. with nodes of different speed). It is well know that standard ring algorithms, with one process per node, are perfectly balanced on homogeneous and dedicated parallel systems [2], but they give poor performance on heterogeneous, possibly non dedicated, resources. In particular, we show as a virtual ring of processes, with data evenly partitioned among processes, allows to obtain very good performance also on highly heterogeneous platforms.

As test application we consider the computation of long- and short-range interactions. In particular, we deal with the computation and histogram of distances (all distances or just those involving neighbouring pairs) in a large set of points in R^m. Such test problem is interesting, since varying the neighbour size we are able to vary the computation to communication ratio.

After the computational analysis of this approach, which gives us both a full computer simulator and some simplified indices of performance, we present and compare some experimental results collected on a heterogeneous platform, namely a cluster of PCs with Windows O.S.

To implement the application, we consider two freely available implementations of MPI for Windows: MPICH NT 1.2.5 and MPICH2 1.0-1. As comparison, we also consider the version for Windows of the older but still popular PVM library (PVM v. 3.4.5).

2. The heterogeneous computing system

Let us consider a heterogeneous parallel system consisting of p nodes, connected by a switched communication network (e.g. Fast Ethernet, Gigabit Ethernet). Our analysis also applies to mixed networks.

Let be s_i the relative speed of node i with respect to a reference machine [3], e.g. the lowest machine in the set. s_i depends both on the processor features and on the application under consideration, and can only be estimated in an approximate way. In particular, it can depend on the amount of local data.

We suppose that the system is dedicated or that the load of each node remains nearly constant during the whole computation. So, time variation of speeds during computation is negligible.

The total relative speed of the system is $S = \sum_i s_i$ and it is also the ideal speed-up.

The speed-up and the global efficiency are [3]:

$$SU = \frac{T^{seq}}{T^{par}} = \sum_i s_i \eta_i, \qquad \eta = \frac{SU}{S} = \frac{\sum_i s_i \eta_i}{S} \tag{1}$$

where: T^{seq} and T^{par} are the execution time on the reference node and the parallel execution time on the heterogeneous system, respectively; $\eta_i, \quad i = 1, \ldots, p$ are the node efficiencies.

The degree of heterogeneity can be expressed in various ways; for ring applications we find useful the index $h = 1 - s_0/\overline{s}$, where $\overline{s}$ is the average relative speed and s_0 denotes the lowest relative speed in the system $(0 \leq h < 1)$.

3. The computational problem

To test the proposed method we employ an application that we originally developed for homogeneous systems [4], and that we subsequently used with various computing platforms. It can be considered a kind of high level benchmark.

Given a set X of N points in R^m, the algorithm carries out the computation and histogram of distances between neighbouring pairs [2,4], i.e. pairs whose distance is less than a predetermined threshold ϵ, expressed as a fraction of the diameter of the pointset. As a particular case, if $\epsilon = 1$, all distances are computed. For the fast search of neighbouring pairs we use the box-assisted approach described in [4], in which an auxiliary m'-dimensional mesh of boxes, with $1/\epsilon$ boxes along each dimension, is used to build linked lists of points falling into the same m'-dim box.

The sequential execution time on the reference node is

$$T^{seq} = \left(\frac{N \cdot (N-1)}{2} \cdot f(\epsilon) \right) \cdot \tau_o(m) \tag{2}$$

where $\tau_o(m)$ is the CPU time to process a pair on the reference node, and $f(\epsilon) \leq 1$ is the fraction of the total pairs processed.

Our application is split into a virtual ring of $q \geq p$ processes, with q_i (logically neighbouring) processes on the i-th node (Fig. 1).

The set of points is evenly partitioned into q subsets X_j, of size $N' = \frac{N}{q}$ and each subset is assigned to a process. The total number of distinct pairs is decomposed in the following way

$$\frac{N \cdot (N-1)}{2} = q \left(\frac{N' \cdot (N'-1)}{2} + \frac{q-1}{2} \cdot N'^2 \right). \tag{3}$$

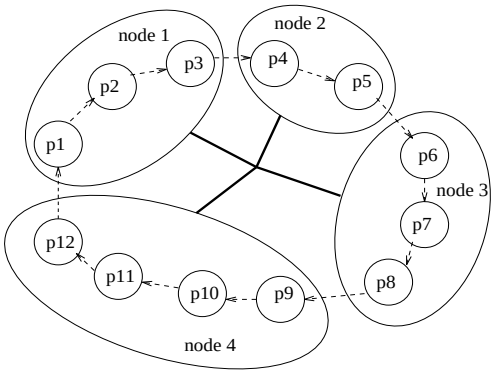

Figure 1. A virtual ring of 12 processes mapped on 4 nodes connected by a switched network.

The computation comprises two phases (Fig. 2): in phase one, which does not need communications, local pairs are processed; in the second phase, which consists of $L = \frac{q-1}{2}$ steps, local points of the generic process are moved forward along the ring in order to process pairs formed by local and visiting points.

Proc$_j$:
compute distances $\leq \epsilon$ in (X_j)
for $l = 1, L$
send data to next process
receive data (X_r) from previous process
compute distances $\leq \epsilon$ in (X_r, X_j)
endfor

Figure 2. Process structure ($1 \leq j \leq q$).

For each process, the CPU times (on the reference node) spent in phase 1 and in each step of phase 2 are:

$$t_1^{cp} = \frac{N'(N'-1)}{2} \cdot f(\epsilon) \cdot \tau_o, \qquad t_2^{cp} = N'^2 \cdot f(\epsilon) \cdot \tau_o. \tag{4}$$

If q is odd the computation is terminated; if q is even a further phase is needed in which $q/2$ processes receive points from their opposite. In the end all the partial histograms are summed. The fraction of total computation carried out in phase 1 $(1/q)$ becomes negligible as q increases.

As ϵ decreases $f(\epsilon)$ decreases, and the computation to communication ratio worsens.

We assume that the various subsets are statistically equivalent, i.e. $f(\epsilon)$ is the same for all subsets, resulting in a constant time to process any pair of subsets.

4. Performance analysis and modeling

The elapsed time on each node is the sum of computation time, context switching overhead, time needed for local activities involved with communications, and idle time (in general due to transmission time and imbalance).

We assume that, for each step in phase 2, the transmission time over the network between nodes i and j can be modeled as:

$$t_{i,j}^{tm} = \alpha_i + \beta_{i,j} \cdot size(N'm) \tag{5}$$

where α_i is the latency, $\beta_{i,j}$ is the communication time per byte, and $size(N'm)$ is the number of bytes to be moved. Similarly, using suitable parameters $\alpha_0, \beta_0, \alpha_{pk}$ and β_{pk}, we can model the intra-processor communication time t_0^{tm}, and the time for packing/unpacking $t^{pk} = t^{upk}$.
t_0^{tm}, t^{pk} and t^{upk} refer to the reference node, and are assumed to vary among nodes as $1/s_i$.

If N' is large it can be convenient to split data into strips of suitable size.

The cumulative elapsed time $T_{i,l}$ on the i-th node at the end of the l-th step is:

$$\begin{aligned} T_{i,0} &= t_i^1 \\ T_{i,l} &= T_{i,l-1} + t_{i,l}, \quad l = 1, \ldots, L \end{aligned} \tag{6}$$

t_i^1 is the elapsed time on the i-th node at the end of the local phase and $t_{i,l}$ is the elapsed time on the i-th node for the l-th step:

$$\begin{aligned} t_i^1 &= q_i \frac{t_1^{cp}}{s_i} + t_i^{cs1} \\ t_{i,l} &= \frac{q_i}{s_i}(t_2^{cp} + t^{pk} + t^{upk}) + t_i^{cs2} + \frac{(q_i - 1)}{s_i} t_0^{tm} + t_{i,l}^{idle} \end{aligned} \tag{7}$$

t_i^{cs1} and t_i^{cs2} denote the time lost due to context switching on the i-th node in the local phase and in one step respectively.

$t_{i,l}^{idle}$ is related to imbalance, and is computed at each step in the following way:

$$\begin{aligned} t_{i,l}^{idle} = \max(\; & T_{i-1,l-1} + \frac{q_{i-1}}{s_{i-1}}(t_2^{cp} + t^{pk}) + \frac{(q_{i-1} - 1)}{s_{i-1}}(t_0^{tm} + t^{upk}) + t_{i-1}^{cs2} + t_{i,j}^{tm} \\ & - T_{i,l-1} - \frac{q_i}{s_i}(t_2^{cp} + t^{pk}) - \frac{(q_i - 1)}{s_i}(t_0^{tm} + t^{upk}) - t_i^{cs2}, \;\; 0 \;). \end{aligned} \tag{8}$$

The parallel execution time is therefore $T^{par} = \max_i T_{i,L}$.

This formulation allows us to implement a computer simulator, whose accuracy depends on the number of overhead sources we consider.

Considering the CPU times at the three levels (step l on node i, node i, whole system) and the corresponding elapsed times, we can obtain [3] the efficiencies $\eta_{i,l}, \eta_i$ and η. The system wide efficiency is related to the node-level quantities by eq. (1).

The imbalance at the step level for the i-th node can be defined as

$$U_{s,i} = t_2^{cp}(q_{i_o}/s_{i_o} - q_i/s_i) \tag{9}$$

where i_o denotes the node with the highest CPU time ($q_{i_o}/s_{i_o} = \max_i(q_i/s_i)$).

Since imbalance is the main factor that limits performance for ring applications on heterogeneous systems, it is interesting to obtain in a approximate way the efficiency loss at the node level due to imbalance as

$$\gamma_i = \frac{U_{s,i}}{t_2^{cp}(q_i/s_i) + U_{s,i}} = 1 - \frac{s_{i_o}}{q_{i_o}} \frac{q_i}{s_i}. \tag{10}$$

Using eq. (1) we obtain the system wide quantity

$$\gamma = 1 - \frac{s_{i_o}}{q_{i_o}} \frac{q}{S}. \tag{11}$$

Following this approach, given an heterogeneous system and given the total number of processes q, the optimal allocation of processes is the one that minimizes $U_{s,i}$. This allocation can be found using dynamic programming [5], or, in a simpler but sub-optimal way, setting $q_i \simeq \frac{s_i}{S} q, \quad i = 1, \ldots, p$, and approximating fractional results to the nearest integer.

5. Experimental and simulated results

5.1. The PC cluster and the message passing environments

The application is tested on 5 different configurations selected in a cluster composed of 7 PCs (see Table 1) with Windows 2000 O.S., connected by switched Fast-Ethernet. The relative speeds s_i are average values since they present some fluctuations both with N' and with ϵ.

Table 1
Description of the various nodes in the cluster; a is the reference node

Node Id.	Type	Memory Size (MB)	s_i
a	PIII 600 MHz	128	1.00
b	PIII 800 MHz	128	1.20
c	PIII 866 MHz	256	1.30
d	PIII 1.3 GHz	256	1.80
e	PIV 1.8 GHz	256	2.07
f	PIV 2.4 GHz	256	3.28
g	PIV 2.8 GHz	512	3.59

To implement the application, we consider two message passing environments: MPICH NT 1.2.5 and MPICH2 1.0-1. They are the versions for Windows of the freely available and portable implementations of MPI (MPICH [6]) and MPI-2 (MPICH2 [7]). As comparison, we also consider the version for Windows of the popular PVM library (PVM v. 3.4.5) [8,9], older but still largely used for its simplicity and effectiveness. We use the Compaq Visual Fortran compiler v.6.6. We performed some preliminary point-to-point communication trials with the three environments, whose results are summarized in Table 2.

Table 2
Comparison of point-to-point communication speeds for the three message passing environments

Environment	$\alpha(\mu s)$	$\beta(ns/byte)$	$\alpha_o(\mu s)$	$\beta_o(ns/byte)$
MPICH2	30-120	90	10-45	11-43
MPICH	38-130	90	4-12	1-6
PVM	5000	300	5000	40-300

It results that communications in PVM are quite poor, since communication speed is limited by the various software overheads at the O.S. and PVM levels, resulting in a high latency and in a sustained rate well below the physical limit; moreover, intra-processor communications are not much faster than inter-processor communications [1].

Communications in MPICH and MPICH2 are significantly more efficient; moreover MPICH2 yields slightly faster inter-processor communications but slower intra-processor communications. Indeed MPICH uses sockets for inter-processor communications and shared memory for intra-processor communications, whereas our implementation of MPICH2 always uses sockets. In order to assure the maximum speed we implement ring communications in MPICH and MPICH2 using the buffered send (BSEND).

5.2. The procedure

We consider different problem sizes and various ϵ values. Points in R^m are obtained from the Henon time series [4] and normalized to the unitary hypercube; we choose $m = 10$, $m' = 2$ and use the euclidean norm. The time per pair measured on the reference node (PIII 600 MHz) is $\tau_o = 0.57\mu s$ and the $f(\epsilon)$ values for our data set are: $f(1) = 1$, $f(2^{-3}) = 0.23$, $f(2^{-6}) = 0.023$, $f(2^{-9}) = 0.0051$.

The trials are executed on dedicated nodes and with a low traffic on the network.

Since our aim is the development of self adaptive applications [10], able to maximize performance for a given hardware configuration, we employ the following procedure.

1) At the beginning we execute on the target system a small instance of the computation, with one process per node, just to evaluate the actual node speeds.

2) Depending on the measured s_i, a suitable number q of ring processes is selected, using eq. (11) and fixing the maximum allowed efficiency loss; for our cluster configurations, 35 processes allow in most case to have $\gamma \leq 0.1$; then the optimal mapping of ring processes to nodes is found [5] and processes are launched.

3) The spawn of the requested number of processes is carried out in PVM using the pvm_spawn routine. In MPICH2 similar routines are available (MPI_comm_spawn and MPI_comm_spawn_multiple), but unfortunately they do not work well with our current release of MPICH2 for Windows. So, for both MPICH and MPICH2 environments, we use a procedure which spawns the optimal number of ring processes by means of $mpirun$ (MPICH) and $mpiexec$ (MPICH2) commands.

This approach assures us that the mapping is optimal, provided the load characteristics of nodes remain nearly constant during computation.

5.3. Analysis of results

The results of the trials are reported in Tables 3,4,5.

An accurate analysis of performance is difficult since various effects interact: for example the variation of processor speeds with the amount of local data, the time spent by the various processes in paging (this time varies with the amount of data of processes), the time for context switching of processes, the actual implementation of inter- and intra-node communications, etc. These effects give in some cases a superlinear speed-up.

However the main results can be summarized as follows:

a) the naive porting (one process per node) gives poor efficiencies whereas the virtual processes approach performs very well (Table 3);

b) as the degree of heterogeneity increases a higher q value is needed, on average, to achieve the same balancing (expressed by $1 - \gamma$ in Table 4);

c) all three message passing environments are able to give good performance; differences are appreciable only when the computation to communication ratio is small and in such situations MPI (both

Table 3
Execution time (in seconds) and measured (η) and simulated (η_s) efficiencies obtained on configuration (a-g) ($S = 14.24$, $h = 0.51$) using MPICH2

N	i ($\epsilon = 2^{-i}$)	T_{seq}	q	T	η	η_s
42 000	0	596.8	7	100.8	0.42	0.46
”	”	”	35	49.1	0.85	0.84
210 000	3	2614	7	316.2	0.58	0.53
”	”	”	35	158.2	1.16	0.96
420 000	3	9554	7	1366	0.49	0.48
”	”	”	35	636.5	1.05	0.88
420 000	6	975.5	7	139.7	0.49	0.49
”	”	”	35	61.2	1.12	0.89
420 000	9	191.9	7	29.5	0.46	0.44
”	”	”	35	17.7	0.76	0.77
840 000	9	751.7	7	117.8	0.45	0.46
”	”	”	35	62.8	0.84	0.82

Table 4
Comparison of three different configurations of 4 nodes with MPICH2 ($N = 105\,000$, $\epsilon = 2^{-3}$, $T_{seq} = 560.6$)

config	S	h	q	$1-\gamma$	T	η	η_s
a,b,c,d	5.30	0.25	15	0.92	112.1	0.94	0.92
a,c,e,g	7.96	0.49	25	0.94	66.3	1.06	0.94
a,e,f,g	9.94	0.60	35	0.96	57.3	0.99	0.96

implementations) outperforms PVM, owing to faster communications (Table 5);
d) MPICH2 is quite slower in the start-up phase (spawn of processes and sending of portions of points to processes);
e) the context switching overhead is always negligible for the typical number of processes per node we consider;
f) experimental and simulated results are in most cases in very good agreement (Tables 3,4,5); the most relevant source of error is probably the fluctuation of speeds with the size of local data.

6. Conclusions

We analyze and model performance of ring algorithms on heterogeneous Windows-based clusters with various message passing environments, with the aim of developing efficient and auto-adaptive applications. The general problem is exemplified considering the computation of long- and short-range interactions.

The algorithm analysis and the experimental results show that the virtual ring approach, with a number of processes much greater than the number of nodes, and with a number of processes in each node matching the node speed, is a feasible technique to exploit a very high fraction of the available power, even for platforms with a high heterogeneity. Depending on ϵ, we are able to process up to millions of high dimensional points ($m = 10$ or greater).

The algorithm is tested on a cluster of PCs connected by switched Fast-Ethernet with Windows

Table 5
Comparison of the three message passing environments (conf=(a,b,e,f,g), $S = 11.14$, $h = 0.55$, $N = 840\,000$, $\epsilon = 2^{-9}$, $T_{seq} = 751.7$)

	MPICH2			MPICH			PVM		
q	T	η	η_s	T	η	η_s	T	η	η_s
5	168.0	0.40	0.38	169.1	0.40	0.38	203.4	0.33	0.38
15	97.2	0.69	0.75	97.3	0.69	0.75	126.5	0.53	0.72
21	86.4	0.78	0.80	83.9	0.80	0.80	108.7	0.62	0.75
25	84.0	0.80	0.78	83.8	0.81	0.78	106.4	0.63	0.71
35	94.9	0.71	0.79	78.2	0.86	0.80	106.3	0.63	0.69

O.S., using three message passing libraries: MPICH2, MPICH and PVM. Whereas some differences in performance depend on the details of the implementation of the message passing library, it turns out that virtual ring algorithms for heterogeneous Windows-based clusters perform in all situations in a very satisfactory way.

The proposed analysis and procedure are quite general, and apply to any ring algorithm on heterogeneous platforms. Detailed models, as the one presented in Sect. 4, which can be very accurate, but rely on parameters which are often known in an approximate way, can be useful to understand in depth the behaviour of the application. However, in practical situations we need robust and auto-adaptive procedures, able to grasp the basic aspects of the computing platform/application.

As future work we will consider the problems arising when the load of nodes varies during computation [11,12]. In this case we need the dynamic migration of processes among the various nodes, according to node load. Another possible approach is to use threads instead of processes.

References

[1] J.K. Hollingsworth, E. Guven, C. Akinlar, Benchmarking a network of PCs running parallel applications, Proc. IEEE Int. Performance, Computing and Communications Conference, IEEE (1998) 1-7.

[2] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, D.W. Walker, Solving Problems on Concurrent Processors, Vol. 1. Prentice-Hall, Englewood Cliffs, NJ (1988).

[3] A. Clematis, A. Corana, Porting regular applications on heterogeneous workstation networks: performance analysis and modeling, Parallel Algorithms and Applications 17 (2002) 205-226.

[4] A. Corana, Parallel computation of the correlation dimension from a time series, Parallel Computing 25 (1999) 639-666.

[5] P. Boulet, J. Dongarra, Y. Robert, F. Vivien, Static tiling for heterogeneous computing platforms, Parallel Computing 25 (1999) 547-568.

[6] MPICH. http://www-unix.mcs.anl.gov/mpi/mpich/

[7] MPICH2. http://www-unix.mcs.anl.gov/mpi/mpich2/

[8] PVM. http://www.csm.ornl.gov/pvm/pvm_home.html

[9] M. Fischer, J. Dongarra, Experiences with Windows 95/NT as a cluster computing platform for parallel computing, J. Parallel and Distributed Computing Practices 2 (1999).

[10] S.S. Vadhiyar, J.J. Dongarra, Self Adaptivity in Grid Computing, Concurrency and Computation: Practice and Experience 17 (2005) 235-257.

[11] O. Sievert, H. Casanova, A simple MPI process swapping architecture for iterative applications, Int. J. High Performance Computer Applications 18 (2004) 341-352.

[12] H. Dail, F. Berman, H. Casanova, A decoupled scheduling approach for Grid application development environments, J. Parallel Distrib. Comput. 63 (2003) 505-524.